ARTICLE **Communicated by Gert Cauwenberghs**

Energy-Efficient Neuromorphic Classifiers

Daniel Martí
daniel.marti@ens.fr
Département d'Études Cognitives, École Normale Supérieure–PSL Research University, 75005 Paris, France; Institut Nationale de la Santé et de la Recherche Médicale, 75005 Paris, France; and Center for Theoretical Neuroscience, Columbia University, College of Physicians and Surgeons, New York, NY 10032, U.S.A.

Mattia Rigotti
mr2666@columbia.edu
IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, U.S.A., and Center for Theoretical Neuroscience, Columbia University, College of Physicians and Surgeons, New York, NY 10032, U.S.A.

Mingoo Seok
ms4415columbia.edu
Department of Electrical Engineering, Columbia University, New York, NY 10027, U.S.A.

Stefano Fusi
sf2237@columbia.edu
Center for Theoretical Neuroscience and Mortimer B. Zuckerman Mind Brain Behavior Institute, Columbia University, College of Physicians and Surgeons, New York, NY 10032, U.S.A.

Neuromorphic engineering combines the architectural and computational principles of systems neuroscience with semiconductor electronics, with the aim of building efficient and compact devices that mimic the synaptic and neural machinery of the brain. The energy consumptions promised by neuromorphic engineering are extremely low, comparable to those of the nervous system. Until now, however, the neuromorphic approach has been restricted to relatively simple circuits and specialized functions, thereby obfuscating a direct comparison of their energy consumption to that used by conventional von Neumann digital machines solving real-world tasks. Here we show that a recent technology developed by IBM can be leveraged to realize neuromorphic circuits that operate as classifiers of complex real-world stimuli. Specifically, we provide a set of general prescriptions to enable the practical implementation of neural architectures that compete with state-of-the-art classifiers. We also show that the energy consumption of these architectures, realized

Neural Computation **28, 2011–2044** (2016)
doi:10.1162/NECO_a_00882

on the IBM chip, is typically two or more orders of magnitude lower than that of conventional digital machines implementing classifiers with comparable performance. Moreover, the spike-based dynamics display a trade-off between integration time and accuracy, which naturally translates into algorithms that can be flexibly deployed for either fast and approximate classifications, or more accurate classifications at the mere expense of longer running times and higher energy costs. This work finally proves that the neuromorphic approach can be efficiently used in real-world applications and has significant advantages over conventional digital devices when energy consumption is considered.

1 Introduction

Recent developments in digital technology and machine learning are enabling computers to perform an increasing number of tasks that were once solely the domain of human expertise, such as recognizing a face in a picture or driving a car in city traffic. These are impressive achievements, but we should keep in mind that the human brain carries out tasks of such complexity using only a small fraction of the energy needed by conventional computers, the difference in energy consumption often being of several orders of magnitude. This suggests that one way to reduce energy consumption is to design machines by taking inspiration from the biological brain, an approach that Carver Mead proposed in the late 1980s (Mead, 1989) and that is known as neuromorphic engineering. Mead's idea was to use very-large-scale integration (VLSI) technology to build electronic circuits that mimic the architecture of the nervous system. The first electronic devices inspired by this concept were analog circuits that exploited the subthreshold properties of transistors to emulate the biophysics of real neurons. Today the term *neuromorphic* refers to any analog, digital, or hybrid VLSI system whose design principles are inspired by those of biological neural systems (Indiveri et al., 2011).

Neuromorphic hardware has convincingly demonstrated its potential for energy efficiency, as proven by devices that consume as little as a few picojoules per spike (Livi & Indiveri, 2009; Rangan, Ghosh, Aparin, & Cauwenberghs, 2010; Chicca, Stefanini, & Indiveri, 2014; McDonnell, Boahen, Ijspeert, & Sejnowski, 2014). Until a few years ago, however, these devices contained a relatively small number of neurons and synapses, and they could typically perform only simple and specialized tasks, making it difficult to directly compare their energy consumption to that of conventional digital machines.

The situation has recently changed with the development of large-scale neuromorphic architectures which contain enough units to potentially perform complex real-world tasks (Benjamin et al., 2014; Painkras et al., 2013; Pfeil et al., 2013; Neftci, Das, Pedroni, Kreutz-Delgado, & Cauwenberghs,

2014; Neftci, Pedroni, Joshi, Al-Shedivat, & Cauwenberghs, 2015; O'Connor, Neil, Liu, Delbruck, & Pfeiffer, 2013; Hussain & Basu, 2016). Our analysis focuses on one of the latest generations of such neuromorphic platforms, the IBM TrueNorth processor recently introduced by Merolla et al. (2014). The energy consumption of TrueNorth was estimated with the implementation of a complex recurrent neural network showing spontaneous activity, and it was shown to be 10^5 times smaller than the energy consumption of an optimized simulator of the exact same network running on a general-purpose microprocessor (Merolla et al., 2014). While this is a huge difference, the comparison could be considered unfair because general-purpose microprocessors have not been designed to simulate neuromorphic systems. To make a fairer comparison, we estimated the energy consumption of the TrueNorth and a conventional digital machine during the execution of a complex task. Specifically, we considered a few standard classification tasks and compared the energy consumption of the TrueNorth with that of state-of-the-art conventional digital devices simulating either a support vector machine (SVM) or a neural network that was functionally equivalent to the one implemented on the TrueNorth. We trained TrueNorth, the SVM, and the neural network to solve the same task, and we compared energy consumptions at equal classification performance. Our analysis shows that our chip-implemented classifier uses two or more orders of magnitude less energy than current digital machines. These results show the deployment of a neuromorphic device able to solve a complex task while meeting the claims of energy efficiency by the neuromorphic engineering community for the last few decades.

2 Results

We chose pattern classification as an example of a complex task because of the availability of well-established benchmarks. A classifier takes an input, like the image of a handwritten character and returns one element out of a set of discrete classes, like the set of digits. To train and evaluate our classifiers, we used three different data sets consisting of images of different complexity (see Figure 1a).

We start by describing the architecture of the classifier that we plan to implement on the neuromorphic chip. The classifier is a feedforward neural network with three layers of neurons, and it can be simulated on a traditional digital computer. We will call this network the *neural classifier* to distinguish it from its final chip implementation, which requires adapting the architecture to the connectivity constraints imposed by the hardware. The neural classifier also differs from the final hardware implementation in two more aspects. First, the neural classifier employs neurons with a continuous activation function, whereas the IBM neuromorphic chip emulates spiking neurons. Second, the chip's synapses have limited precision, whereas the neural classifier could take advantage of the double precision

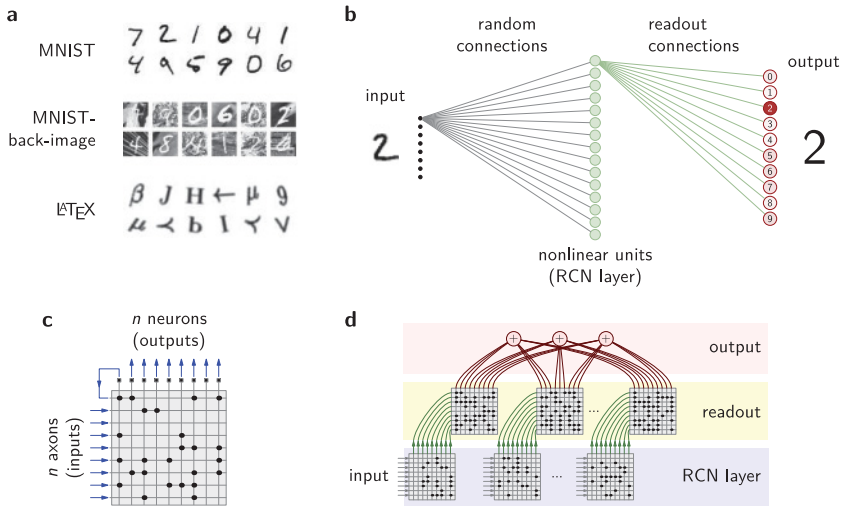

Figure 1: Data sets, architecture of the classifier, architecture of a single core, and chip implementation. (a) Samples of the three data sets used to evaluate the performance of our classifier. MNIST contains handwritten digits (10 classes); MNIST-back-image contains the digits of the MNIST data set on a background patch extracted randomly from a set of 20 images downloaded from the Internet; LaTeX contains distorted versions of 293 characters used in the LaTeX document preparation system. For more details about the data sets, see the appendix. (b) Architecture of the neural network classifier. The images to classify are preprocessed and represented as patterns of activity of a population of N_{in} input neurons (left, black dots). These input neurons send random projections to a second layer of N randomly connected neurons (RCNs) (green circles), which transform nonlinearly their synaptic inputs into firing activity. The activity of the RCNs is then read out by a third layer of neurons, each trained to respond to only one class (red circles). (c) Architecture of a single core in the chip. Horizontal lines represent inputs, which are provided by the axons of neurons projecting to the core. Vertical lines represent the dendrites of the neurons in the core. There is only one dendrite per neuron. Active synapses are shown as dots in a particular axon-dendrite junction. The synaptic input collected by the dendrites is integrated and transduced into spike activity at the soma (filled squares on top), and the resulting spikes are sent via the neuron's axon to a particular input line, not necessarily on the same core. Blue lines represent the flow of input and output signals. The panel includes an example of internal connection: the upmost axon carries the output activity of the leftmost neuron in the core (other connections are left unspecified). (d) Implementation of the neural network classifier in a chip with connectivity constraints. The input is fed into all the cores in the RCN layer (shaded blue), whose neurons project to the input lines of readout cores (shaded yellow) in a one-to-one manner (green curves). The outputs of the readout units are combined together offline to generate the response of the output neuron (shaded red). See the text for the description of the different modules.

floating point representations used in digital computers. Despite these differences, the functionality of the neural classifier and its final chip implementation is approximately the same, as we show. We will explain the procedures for adapting the architecture of the neural classifier into its chip implementation as a contribution in its own right, since they can be used to implement generic neural systems on other hardware substrates.

2.1 Architecture of the Neural Classifier. Figure 1b illustrates the three-layer neural classifier. The first layer encodes the preprocessed input and projects to the neurons in the intermediate layer through connections with random weights. We aptly call the neurons in the intermediate layer randomly connected neurons (RCNs). Each such RCN receives a synaptic current given by a randomly weighted sum of the inputs and transforms it into activation levels in a nonlinear way—in our case, through a linear rectification function: $f(x) = x$ if $x > 0$, and 0 otherwise. The combination of a random mixing of the inputs and a nonlinear input-output transformation efficiently expands the dimensionality of the resulting signal (see Jaeger & Haas, 2004; Buonomano & Maass, 2009; Barak, Rigotti, & Fusi, 2013), thereby increasing the chances that downstream neurons can discriminate signals belonging to distinct classes. This discrimination is carried out by a set of output units in the last layer, which compute a weighted sum of the RCNs activity with a set of weights trained so that each output unit responds to one separate class (one-versus-all code). Details are given in the Appendix. Once the network is trained, a class is assigned to each input pattern according to which output unit exhibits the highest activation.

2.2 Chip Implementation of the Neural Classifier. We implemented the neural classifier on the IBM neuromorphic chip described in Arthur et al. (2012) and Merolla et al. (2014). The first step for converting the abstract neural classifier into an explicit chip implementation is to transform the input patterns into a format compatible with the spike-based coding of the TrueNorth system. For this, we simply employ a firing rate code and convert the integer value of every input component into a spike train with a proportional number of spikes, a prescription that is commonly used in neurocomputational models such as the neural engineering framework (Eliasmith et al., 2012). Specifically, input patterns are preprocessed and formatted into 256-dimensional vectors representing the firing activity of the input layer (identically to the preprocessing step applied in the neural classifier; see the appendix). This vector of activities is then used to generate 256 regular-firing spike trains that are fed into a set of cores with random and sparse connectivity. This set of cores constitutes the RCN layer. As in the neural classifier, all neurons in the RCN layer receive synaptic inputs that consist of randomly weighted combinations of the input and transform their synaptic inputs into firing activity according to a nonlinear function. On the chip, this function is given by the neuronal current-to-rate transduction, which approximates a linear-rectification function (Fusi & Mattia, 1999).

We also face the problem of mapping the original connectivity matrix on the chip. Discriminating the inputs coming from the RCN layer requires each output unit to read from the whole layer of RCNs, which in our implementation can contain as many as 2^{14} neurons. Moreover, all readout connections have to be set at the weights computed by the training procedure. These requirements exceed the constraints set by the chip design in terms of the maximal number of both incoming and outgoing connections per neuron, as well as the resolution and the freedom with which synaptic weights can be set. We next present a set of prescriptions that will allow us to circumvent these limitations and successfully instantiate our neural classifier on the IBM system. The prescriptions are described in relation to the TrueNorth architecture, but the types of constraints they solve are shared by any physical implementations of neural systems, whether biological or electronic. It is therefore instructive to discuss in some detail the constraints and the prescriptions to overcome them, as they can be easily extended to other settings.

2.2.1 Constraints on Connectivity. The IBM chip is organized in cores, and each core contains 256 integrate-and-fire neurons and 256 input lines. The intersections between the dendrites of neurons and the input lines form a crossbar matrix of programmable synapses (see Figure 1c). Each neuron can connect to other neurons by projecting its axon (output) to a single input line at the same core or at a different core. With this hardware design, the maximum number of incoming connections per neuron, or *fan in*, is 256. Likewise, the maximal number of outgoing connections per neuron, or *fan out*, is 256, with the additional constraint that each outgoing connection is restricted to target neurons of a same core.

2.2.2 Constraints on Synaptic Weight Precision. Synapses can be inactive or active, and the weight of an active synapse is selected from a set of four values available to each neuron. The values available are signed integers of 9-bit precision and may differ from neuron to neuron. Which of the four values is assigned to an active synapse depends on the input line: all synapses on the same input line are assigned an index that determines which of the four values is taken by each synapse (e.g., if the index assigned to an input line is 2, all synapses on the input line take the second value of the set of four available synaptic weights assigned to each neuron).

2.2.3 Prescription to Overcome the Constraints on Connectivity. In our neural classifier, each unit in the input layer has to project to N RCNs, which is typically much larger than the fan-out limit of the chip. This limitation can be easily circumvented by cloning the input layer as many times as necessary to cover all the projections to RCN neurons (see Figure 2a). The opposite problem occurs at the readout layer: each output unit must be innervated from many more neurons than the fan-in limit. This problem can

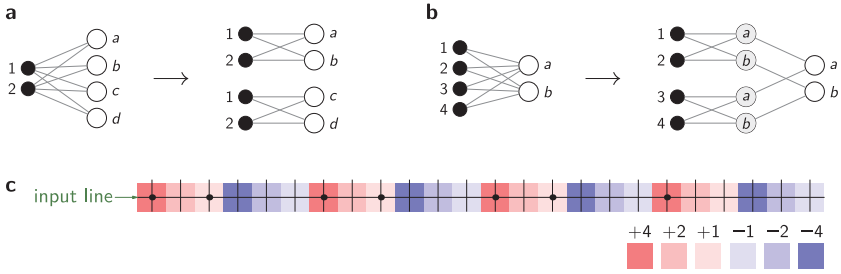

Figure 2: Architectural prescriptions to overcome connectivity constraints. (a) Example of replication of inputs to circumvent fan-out constraints. We want to implement a network with two input units (filled circles, identified with numbers) and four output units (empty circles, identified with letters) on a support with fan-out limited to two (note that this example is chosen for illustrative purposes; for TrueNorth, the fan-out is one output line, which targets 256 neurons. In both cases, the strategy to deal with fan-out constraints is the same.) The original network violates the fan-out constraint at the input layer, but it can be still implemented on hardware by replicating (cloning) the input layer as many times as necessary to innervate all output units. Of course, this prescription has a cost in terms of total number of neurons, which can be significantly larger than in the original architecture. (b) Example of partial sum of outputs to circumvent fan-in constraints. We want to implement a network of four input units and two output units on a support with fan-in limited to two. Although the original network violates the fan-in constraint at the output layer, we can implement the network on constrained hardware by introducing an intermediate layer of units that evaluate the partial contributions to the output. For this to work, the units in the intermediate layer have to operate in a linear regime. (c) Representation of a single synaptic weight on a set of synapses. An input line (horizontal) crosses the set of 24 dendrites that we assign to a given synaptic weight. The dendrites are subdivided in four groups of six synaptic contacts each. Under each axon-dendrite contact is a square that indicates the potential synaptic strength at the site (see the color map at the bottom right). Only the contacts marked with a dot are active. In this case the synaptic value instantiated is 19. See text for details.

be solved by splitting the innervations to the output units so that the fan-in limit is respected, and then have a set of intermediate units compute in parallel different portions of the output (see Figure 2b, intermediate layer). The intermediate layer therefore computes a set of indepedendent "partial outputs," which can then be summed by a downstream neuron whose activity will be the same as that of the original output unit (see Figure 2b, rightmost layer). If the total number of the partial inputs exceeds the fan-in limit, we can iterate the prescription by introducing additional intermediate layers. The number of additional layers scales only logarithmically with the

number of RCNs. For simplicity, we did not implement this tree on chip, and we summed off-chip the partial inputs represented by the firing activity of readout neurons.

Notice that this prescription requires readout neurons to respond approximately linearly to their inputs, a configuration that can be easily achieved by tuning readout neurons to operate in the linear regime of their current-to-rate transduction function (the regime in which their average input current is positive; Fusi & Mattia, 1999). An integrate-and-fire neuron operating in this regime basically implements delta-sigma modulation, which is used to transform an analog signal (in our case, the total synaptic current) into a stream of pulses. Notice that the linearity prescription strongly relies on the assumption that information is encoded in the firing rates of neurons; if the spiking inputs happen to be highly synchronized and synchronization encodes important information, this approach will not work.

2.2.4 Prescription to Overcome the Constraints on Synaptic Weight Precision. Reducing the weight precision after learning usually causes only moderate drops in classification performance. For example, the number of random uncorrelated patterns that can be learned by a classifier does not drop substantially by reducing the number of synaptic states, even when there are only two of them (Sompolinsky, 1986). In contrast, the drop in performance is catastrophically larger when the weight precision is limited during learning (Amit & Fusi, 1994; Fusi, 2002), and in some situations the learning problem may even become NP-complete (Garey & Johnson, 1979). In our case, readout weights were first determined off-chip, using digital conventional computers that operate on double-precision floating point numbers (64 bits) and then quantized on an integer scale for the chip implementation. The performance drop is almost negligible for a sufficient number of synaptic levels. Readout weights were quantized on an integer scale between -28 and 28. Each quantized weight was then implemented as the sum of four groups of six synaptic contacts each, where each contact in the group can be either inactive (value 0) or activated at one of the six values: $\pm 1, \pm 2, \pm 4$ (see Figure 2c).

There are multiple ways to express the quantized weight in this setting. The value 19, for instance, can be decomposed as $(4+1)+(4+1)+(4+1)+(4)$ or $(2)+(4+2)+(4+2)+(4+1)$. We solved this multiplicity by choosing the decomposition that is closest to a balanced assignment of the weights across the four groups (e.g., $19 = (4+1)+(4+1)+(4+1)+(4)$). This strategy requires that each original synapse be represented by 24 synapses. We implemented this strategy by replicating each readout neuron 24 times and distributing each original weight across 24 different dendritic trees. These synaptic inputs are then summed together by the offline summation of all readout neuron activities that correspond to the partial inputs to a specific output unit (see the appendix for details). A similar strategy can be used to implement networks with synaptic weights

that have an even larger number of levels; in that case, the number of additional synapses would scale only logarithmically with the total number of levels. It is crucial, however, to limit individual synapses to low values in order to avoid synchronization between neurons. This is why we limited to four the maximum synaptic value of individual synapses in the chip.

2.3 Classification Performance and Speed-Accuracy Trade-Off. Our neuromorphic classifier implemented on the TrueNorth chip was emulated on a simulator developed by IBM. Because the TrueNorth chip is entirely digital, the simulator reproduces exactly the behavior of the chip (Arthur et al., 2012). In Figure 3a we show the dynamics of two typical runs of the simulator classifying images from the MNIST-back-image data set. Upon image presentation, the RCNs in the intermediate layer start integrating the input signal (not shown), and a few tens of milliseconds later, they start emitting spikes, which are passed to the readout neurons. The figure shows the total number of spikes emitted by the readout neurons since input activation, after subtracting the overall activity trend caused by baseline activity.

For simple classifications, in which the input is easily recognizable, the readout neuron associated with the correct class is activated in less than 100 ms (see Figure 3a, top). More difficult cases require the integration of spikes over longer time intervals, because the average synaptic inputs received by different readout neurons can be very similar (see Figure 3a, bottom). This suggests that the performance of the classifier, as measured by the classification accuracy on the test set, should improve with longer integration intervals. This trade-off between speed and performance is illustrated in Figure 3b, which shows the classification performance versus elapsed time for the MNIST and MNIST-back-image data sets. The performance increases monotonically with time until it saturates in about half a second, with a highest performance of 97.27% for MNIST (98.2% with 10-fold bagging) and 77.30% for MNIST-back-image. These performances are not too far from the best classification results achieved so far: 99.06% for MNIST (using maxout networks on the permutation invariant version of the MNIST data set, which does not exploit any prior knowledge about the two-dimensional structure of the patterns; Goodfellow, Warde-Farley, Mirza, Courville, & Bengio, 2013) and 77.39% for MNIST-back-image (with support vector classifiers; Cho & Saul, 2010. Although methods combining deep nets, feature learning, and feature selection can achieve performances as high as 87.75%; Sohn, Zhou, Lee, & Lee, 2013).

2.4 Energy Consumption Estimates. The energy consumption of TrueNorth depends on the spike rate, the number of active synapses, and the average distance traveled by spikes. There is also a baseline energy consumption that depends on the number of used cores (see section D in the appendix for more details). We will now report our estimates of the energy

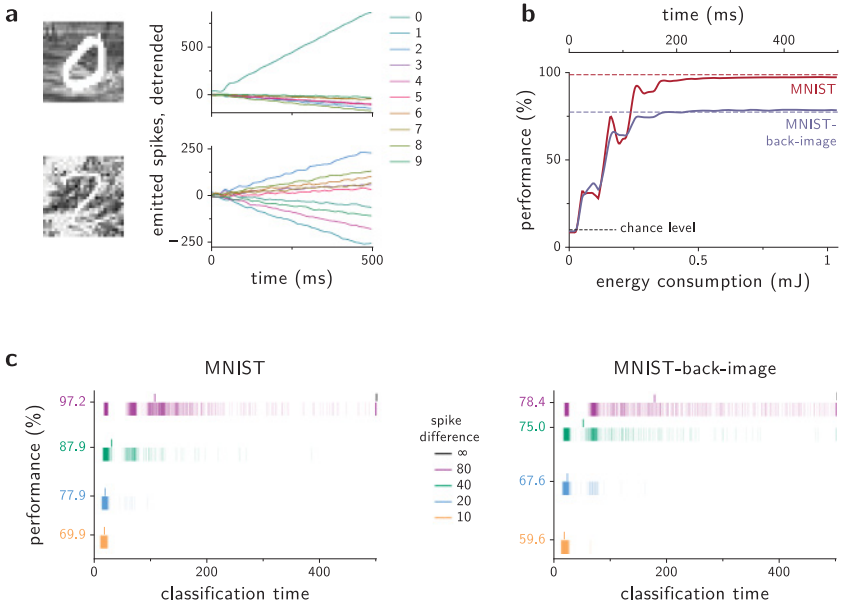

Figure 3: The neuromorphic classifier in action. (a) Spikes emitted by readout neurons during an easy (top) and a difficult (bottom) classification, after removing the trend caused by the intrinsic constant currents. Each curve corresponds to the readout output associated with the digit indicated by the color code. Samples are drawn from the MNIST-back-image data set. (b) Classification accuracy as a function of classification time (i.e., the time over which spikes are integrated) and energy. The accuracy is estimated from the first 1000 test samples of the MNIST (red) and MNIST-back-image (blue) data sets, and its time evolution is smoothed with a causal moving average filter 30 ms long. Each dashed horizontal line indicates the best test error achieved with support vector classifiers for a given data set, based on the evaluation of the whole test set. (c) Classification times for different thresholds in spike difference (as indicated in the legend) for the MNIST and MNIST-back-image data sets. For each threshold we plot all classification times (thin lines) as well as the sample mean (shorter, thicker ticks on top). The performances associated with each threshold are indicated at the y-axis. When the threshold in spike difference is infinite (black), the classification outcome is assessed at $t = 500$ ms (i.e., there is no stopping criterion). In all panels the chip uses $N = 16{,}384$ RCNs.

consumption of TrueNorth and compare them to those of a conventional digital processor performing the same classification task and achieving approximately the same performance. In both cases, we estimate the energy consumption by simulating the chip during a classification. This approach allows us to consider only the energy expended by core computational units

and ignore everything else—for example, the energy costs associated with file access. Unfortunately, the energy estimates based on simulators can be rather inaccurate, in particular for conventional digital processors, for which inaccuracies can change the final estimate by a factor as big as 2 or 3 (Xi, Jacobson, Bose, Wei, & Brooks, 2015; Butko, Garibotti, Ost, & Sassatelli, 2012). It is important to remember, however, that these inaccuracies are relatively small compared to the overall gap in energy consumption found between TrueNorth and conventional digital processors, and therefore they do not affect the main conclusions of our analysis.

2.5 Energy-Speed-Accuracy Trade-Off. Accuracy has a cost in terms of energy because longer integration times entail more emitted spikes per classification and larger baseline energy costs, which in our case is the dominant contribution to the total energy consumption. We estimated the energy consumption as described in section D in the appendix, and we found that the energy per classification never exceeds 1 mJ for our network configuration. With the energy needed to keep a 100 W light bulb lit for 1 s, one could perform 10^5 classifications, which is equivalent to around one classification per second uninterruptedly for almost one day. Notice that this estimate is based on a classification that lasts 0.5 s and therefore does not take into account the fact that most patterns are correctly classified in a significantly shorter time (see Figure 3a, top). We can largely reduce the average energy consumption by stopping the classifier as soon as one of the output units is significantly more active than the others. To implement this early stopping method, we monitored the cumulative activity of each output unit by counting all the spikes emitted by the corresponding readout neurons, and we stopped the classification when the accumulated activity of the leading unit exceeded that of the second leading unit by some threshold. The decision was the class associated with the leading output unit.

In Figure 3c we show the performance and the corresponding classification times for several thresholds. Low thresholds allow for faster yet less accurate classifications. In both the MNIST and MNIST-back-image data sets, the patterns that require long classifications times are rare. While the performance barely changes for large enough thresholds, the average classification time can be substantially reduced by lowering the threshold. For example, for the MNIST data set, the classification time drops by a factor of 5 (from 500 ms to 100 ms) and so does the energy consumption (from 1 mJ to 0.2 mJ). Faster classifications are also possible by increasing either the average firing rate or the total number of RCNs, both of which entail an increase in energy consumption, which might be partially or entirely compensated by the shortening of classification time. These expedients will speed up the integration of spike counts, and as a result, the output class will be determined faster.

In all cases, both the energy cost and the classification performance increase with the total number of emitted spikes or, equivalently, with

integration time (keeping the average firing rate fixed). This is a simple form of a more general energy-speed-accuracy trade-off that has been described in several biological information processing systems (see Lan, Sartori, Neumann, Sourjik, & Tu, 2012), and that can confer great functional flexibility to our classifier. One advantage of basing the computation on a temporal accumulation of spikes is that the classifier can be interrupted at any time at the cost of reduced performance, but without compromising its function. This is in stark contrast to some conventional clock-based centralized architectures whose mode of computation crucially relies on the completion of entire monolithic sets of instructions. We can then envisage scenarios where a spiking-based chip implementation of our classifier is required to flexibly switch between precise long-latency classifications (like those involving the identification of targets of interest) and rapid responses of limited accuracy (like the quick avoidance of imminent danger).

Notice that both the simulated and implemented networks, although entirely feedforward, exhibit complex dynamics leading to classification times that depend on the difficulty associated with the input. This is because neurons are spiking and the final decision requires some sort of accumulation of evidence. When a stimulus is ambiguous, the units representing the different decisions receive similar inputs and the competition becomes harder and longer. This type of behavior is also observed in human brains (Tang et al., 2014).

We now focus on the comparison of energy consumption and performance between the neuromorphic classifier and more conventional digital machines.

2.6 Energy Consumption and Performance: Comparison with Conventional Digital Machines. We compared both the classification performance and the energy consumption of TrueNorth to those obtained with conventional digital machines. We considered two main types of conventional digital machine: a digital implementation of several support vector machines (SVMs) and a simulation of our original neural classifier (see Figure 1). We will refer to the latter as the digital neural classifier. SVMs offer a reasonable comparison because they are among the most successful and widespread techniques for solving machine-learning problems involving classification and regression (Boser, Guyon, & Vapnik, 1992; Cortes & Vapnik, 1995; Vapnik, Golowich, & Smola, 1997), and because they can be efficiently implemented on digital machines. The comparison with the digital neural classifier, on the other hand, allows us to compare circuits that share the same basic architectural design but differ in their physical substrate (neuromorphic or conventional). With this comparison, we can tell apart the features brought about by the circuit design from those inherent to the neuromorphic implementation. It is important to stress that we simulated the original neural classifier, which on digital machines is not subject to any architectural constraint (fan-in, fan-out, and limited synaptic precision).

This means in particular that the number of neurons of the digital neural classifier is typically much smaller than on TrueNorth. For this reason, it was not obvious a priori that TrueNorth would be more energy efficient, even if it used hardware that was specifically designed to implement neural networks.

To better understand how the energy consumption scales with the complexity of the classification problem, it is useful to summarize how SVMs work. After training, SVMs classify an input pattern according to its similarity to a set of templates, called the support vectors, which are determined by the learning algorithm to define the boundaries between classes. The similarity is expressed in terms of the scalar product between the input vectors and the support vectors. As argued above, we can improve classification performance by embedding the input vectors in a higher-dimensional space before classifying them. In this case, SVMs evaluate similarities by computing classical scalar products in the higher-dimensional space. One of the appealing properties of SVMs is that there is no need to compute explicitly the transformation of inputs into high-dimensional representations. Indeed, one can skip this step and compute directly the scalar product between the transformed vectors and templates, provided that one knows how the distances are distorted by the transformation. This is known as the kernel trick because the similarities in a high-dimensional space can be computed and optimized over with a kernel function applied to the inputs. Interestingly, the kernel associated with the transformation induced by the RCNs of our neural classifier can be computed explicitly in the limit of a large number of RCNs (Cho & Saul, 2010). This is the kernel we used to compare the performance of SVMs against that of our neural classifier.

Unfortunately, classifying a test input by computing its similarity to all support vectors becomes unwieldy and computationally inefficient for large data sets, as the number of support vectors typically scales linearly with the size of the training set in many estimation problems (Steinwart & Christmann, 2008). This means that the number of operations to perform, and hence the energy consumption per classification, also scales with the size of the training set, thereby making SVMs and kernel methods both computationally and energetically expensive in large-scale tasks. In contrast, our neural network algorithm evaluates a test sample by means of the transformation carried out by the RCNs. If the RCN layer comprises N neurons and the input dimension is N_{in}, evaluating the output of a test sample requires $O(N_{\text{in}} \cdot N)$ synaptic events. Thus, for large sample sizes, evaluating a test sample in the network requires far fewer operations than when using the kernel trick, because the number is effectively independent of the size of training set (Rahimi & Recht, 2008; Le, Sarlós, & Smola, 2013). Systems such as ours may therefore display considerable energy advantages over SVMs when data sets are large.

In Figure 4 we compare the energy consumption and performance of TrueNorth to those of an SVM and the neural classifier implemented on

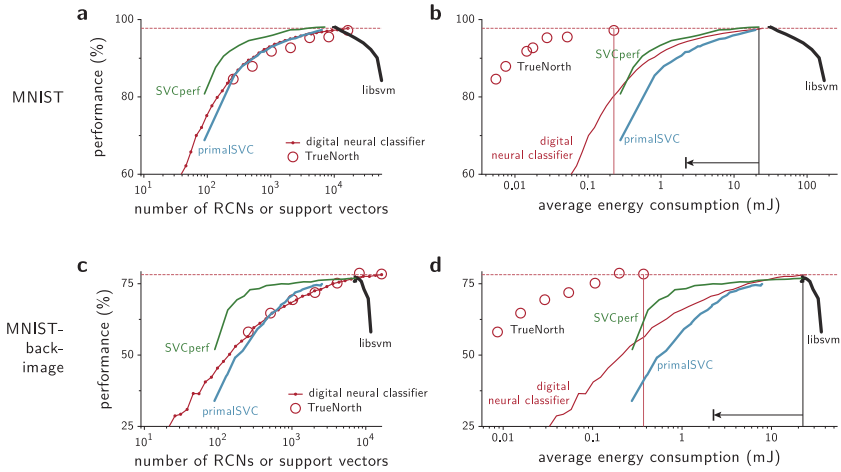

Figure 4: Energy-accuracy trade-off. (a) Dependence of the classification accuracy on the number of randomly connected neurons (RCNs) in the neural classifiers and on the number of support vectors (SVs) in the SVCs, for the MNIST data set. The neural classifiers comprise the implementation on TrueNorth and the nonspiking neural classifier simulated on a digital machine, briefly indicated as "digital neural classifier." We consider three different implementations of support vector classifiers (legend code: SVC, `libsvm`; rSVC, reduced primal; SVC; SVCperf, cutting plane subspace pursuit). The algorithms rSVC, SVCperf minimize the number of support vectors (SVs). For the standard algorithm (libsvm), the number of SVs used can go beyond the optimal value by reducing sufficiently the soft-margin parameter and pushing the classifier to overfit the data. For reference, we indicate the best performance achieved by the digital neural classifier with a horizontal red dashed line. (b) Dependence of the classification accuracy on the energy consumption. For TrueNorth the energy consumption is based on the average time it takes to perform a classification (see Figure 3c). The thin red vertical line is drawn in correspondence of the best performance for TrueNorth. The same performance is achieved by the best SVM implementation where the vertical black line is drawn. The horizontal arrow indicates the reduction in energy consumption that would be attained if the efficiency of digital machines reached the theoretical lower bound estimated by Hasler and Marr (2013). The relationship between the number of SVs and energy consumption was determined by simulating the single-core configuration of the Intel i7 chip running a program that implements an SVM at test time. (c, d) Same as panels a and b but for the MNIST-back-image data set.

a conventional digital machine. More specifically, we estimated the energy expenditure of a digital SVM and the digital neural classifier using a simulator that mimics the single-core configuration of the Intel i7 processor, which is the digital machine with best energy performance among

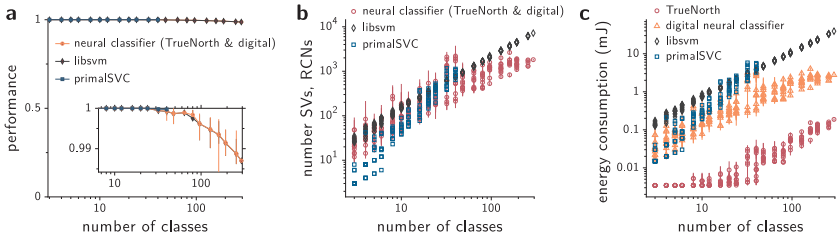

Figure 5: Dependence of the energy consumption on the number of classes. (a) Classification accuracy for TrueNorth, the digital neural classifier, and two SVM algorithms, as a function of the number C of classes for the LaTeX data set. The parameters of the different classifiers are tuned to have approximately the same classification accuracy. (b) Number of support vectors and RCNs as a function of the number of classes for the LaTeX data set. Given a number C of classes, every point in the plot is obtained by training a given classifier on C randomly sampled classes among the 293 available ones. This procedure is repeated 10 times for every value of C and every type of classifier. Each data point associated with the neural classifiers was in turn estimated from a sample of 10 realizations of the random connections (squares indicate sample means; error bars indicate the 0.1 and 0.9 fractile of the sample). (c) As in panel b, but energy consumption as a function of the number of classes. Here we also include the energy consumption of a digital implementation of the neural classifier (legend: digital neural classifier).

those we simulated (see sections 3 and, in the appendix E). The energy cost per support vector per pattern was estimated to be around 3.1 μJ, a quantity that is not far above what is considered as a lower bound on energy consumption for digital machines (Hasler & Marr, 2013). For both the TrueNorth and the digital neural classifier, we progressively increased the performance by increasing the number of RCNs. For the digital SVM, we controlled the performance level by varying the number of support vectors (see Figures 5a and 5c) with the help of three different algorithms that minimize the number of support vectors and hence the energy consumption (for more details, see the caption of Figure 4 and the appendix). The energy consumption of the TrueNorth was estimated in both the early stopping condition, described in the previous section, and in the case in which the classification time was fixed at 500 ms (see Figure 8 in the appendix). Notice that the digital neural classifier carries out its classification by updating all neurons in one step, and it does not exhibit any dynamics because its neurons do not spike. In all the cases that we considered, the energy consumption was significantly lower for TrueNorth, being in the early stopping condition approximately two orders of magnitude smaller for both the MNIST and the MNIST-back-image datasets, while still achieving comparable maximal performances (see Figures 5b and 5d). Interestingly the energy

consumption for the digital neural classifiers and for the SVMs was comparable for both data sets.

2.7 Scalability. The MNIST data set has only 10 output classes. We wondered whether the advantage of TrueNorth in terms of energy consumption is preserved when the number of classes increases and the classification task becomes more complex. To study how the energy consumption scales with the number of classes, we used the LaTeX data set, which contains 293 classes of distorted characters. We progressively increased the number of classes to learn and classify, and we studied the performance and the energy consumption of both the digital implementation of the SVM and the neuromorphic classifier. Specifically, given a number of classes that was varied between 2 and 293, we selected a random subset of all the available classes, and we trained both the SVM and the neural classifier on the same subset. The results were averaged over 10 repetitions, each with a different sample of output classes.

To make a meaningful comparison between the the energy consumed by an SVM, the digital neural classifier and TrueNorth, we equalized all the classification accuracies as follows. For each classification problem, we varied the margin penalty parameter of the standard SVC using grid search and picked the best performance achieved. We then varied the relevant parameters of the other two classifiers so that their classification accuracy matched or exceeded the accuracy of the standard SVC. Specifically, we progressively increased the number of basis functions (in the primalSVC method) and the number of RCNs (in the neural classifier) until both reached the target performance. For each classification problem we averaged over 10 realizations of the random projections of the neural classifier.

The results are summarized in Figure 5. The energy consumption is about two orders of magnitude larger for the SVM throughout the entire range of variation of the number of classes we studied, although for a very small number of classes (two or three), the advantage of TrueNorth strongly reduces, most likely because the algorithms to minimize the number of SVs work best when the number of classes is small. Overall, the energy advantage of TrueNorth over the SVMs implemented on conventional digital machines is maintained also for more complex tasks that involve a larger number of classes.

At this point, it is worth discussing the expected scaling for a growing number of classes. Consider the case of generic C classes' multiclass problems, solved by reducing the problem into multiple binary SVMs. In the one-versus-all reduction scheme, each binary classifier is trained to respond to exactly one of the C classes, and hence C SVMs are required. In general, each binary SVM uses a different set of support vectors, although quite a few support vectors are typically shared by two or more binary SVMs. Let N_{SV} be the total number of unique support vectors used across all SVMs. Classifying a test pattern requires computing the scalar products

between the test pattern and all N_{SV} support vectors. Since each such scalar product requires N_{in} multiplications and sums, this step involves $N_{in} \cdot N_{SV}$ operations. The resulting N_{SV} scalar products should then be multiplied by the coefficients corresponding to each binary SVMs. This additional step requires on the order of CN_{SV}. If N_{SV} scales linearly with C, as in the cases we analyzed, then the total energy E will scale as

$$E \sim N_{in}C + C^2.$$

When C is small compared to N_{in}, the first term dominates, and the expected scaling is linear. However, for $C > N_{in}$, the scaling is expected to be at least quadratic. It can grow more rapidly if the support vectors are different for different classifiers.

Interestingly, the expected scaling for the neural network classifiers implemented on TrueNorth is the same. The energy consumption mostly depends on the number of cores used, which is proportional to NC, the number of RCNs multiplied by the number of classes. To see this dependence, note that each core can receive up to 256 inputs, and therefore the total number of cores has to be proportional to $\lceil N/256 \rceil$, with $\lceil \cdot \rceil$ denoting the ceiling function. In turn, the output lines of these RCNs have to project to the readout layer, whose size is proportional to the number of classes. The total number of cores therefore grows as NC. In the cases we analyzed, N depends linearly on the number of classes, and hence the energy depends quadratically on C, as in the case of the SVMs when C is large enough. Notice that there is a second term that also scales quadratically with C. This term comes from the need to replicate the RCNs C times due to the limited fan-out of the RCNs. Again, under the assumption of $N \sim C$, this term also will scale quadratically with C.

Finally, in the case of the digital neural classifier, we also obtain a similar scaling. In this case, the energy consumption depends on the number of operations that are needed to compute the activation of all the RCNs and the output units. This number is proportional to $N_{in} \cdot N + NC$. The second term again scales as C^2 when one considers that the number of RCNs scales linearly with C.

Given that the scaling with the number of classes is basically the same for TrueNorth, the digital neural classifier, and for the SVMs, it is not unreasonable to hypothesize that the energy consumption advantage of neuromorphic implementations like TrueNorth would also be preserved for a much larger number of classes.

3 Discussion

Our results indicate that neuromorphic devices are mature enough to perform a real-world machine learning task and achieve a performance that is

comparable to that obtained with state-of-the-art conventional devices with von Neumann architecture, all just by using a tiny fraction of their energy. Our conclusions are based on a few significant tests, based on a comparison limited to our neuromorphic classifier and a few digital implementations of SVMs. This clearly restricts the generality of our results and does not preclude situations in which the advantage of the neuromorphic approach might be less prominent. Notwithstanding very recent work on an alternative spiking neuromorphic hardware, the SpiNNaker platform (Painkras et al., 2013), has demonstrated the successful implementation of a different machine learning architecture, also with good performance on MNIST and remarkably low power consumption (Stromatias et al., 2015). One of the advantages of the TrueNorth chip is that it still offers the potential to be interfaced through a spike-oriented asynchronous bus, with a neuromorphic sensor (e.g., a silicon retina; Delbruck & Lang, 2013), thereby keeping up with the idea that neuromorphic neural networks may ultimately be embedded in small and low-power systems for robotic applications.

In any case, the main merit of our study is to offer a solid comparison with implementations on current conventional digital platforms that are energy efficient themselves. In particular, the algorithm we used on conventional digital machines involves only multiplications between matrices and vectors, the efficiency of which has been dramatically increased in the last decades thanks to optimized parallelization. Furthermore, not only did we try to match the classification performance of the competitors; we also considered two additional SVM algorithms that minimize the number of support vectors and therefore the final number of operations. Other choices for SVM algorithms would certainly lead to different estimates for energy consumption, but it is rather unlikely that they would change across two orders of magnitude. It is possible that full custom digital machines would be more energy efficient, but it is hard to imagine that they would break the predicted energy wall discussed in Hasler and Marr (2013). If this assumption is right, neuromorphic hardware would always be more efficient when performing the type of tasks that we considered. Moreover, analog neuromorphic VLSI or unreliable digital technologies might allow for a further reduction of energy consumption, probably by another order of magnitude (Chicca et al., 2014; Arthur & Boahen, 2011; Han & Orshansky, 2013). The current energy consumption levels achieved by analog systems are very close to those of biological brains in terms of energy per spike, although many of these systems are relatively small, and it is unclear whether they can ever be extended to brain-scale architectures.

Other promising hardware solutions have been used to implement neural network algorithms that are able to solve challenging real-world tasks. An example is NeuFlow, implemented with both FPGAs (Farabet et al., 2011) and as a full custom chip (Pham et al., 2012). The NeuFlow architecture was designed to implement convolutional networks for visual recognition. The FPGA implementation uses approximately 800 mJ per classification in

a complex street scene parsing task. Comparing the energy consumption of NeuFlow and our classifier implemented on TrueNorth would be difficult, as the NeuFlow is a network that is significantly more complex than the one we considered. Indeed, NeuFlow is a network with eight layers, which contains many more neurons, being designed to process larger images.

More generally, the FPGA approach is interesting and extremely promising, and it should be considered in future benchmarks on energy consumption. FPGA architectures that are massively parallel are potentially more energy efficient than the digital devices we considered. However, it is important to consider that current FPGA devices still suffer from two limitations. The first is that both FPGAs and microprocessors might need to access external memory when a von Neumann architecture is considered (Gupta, Agrawal, Gopalakrishnan, & Narayanan, 2015). In fact, high-end FPGA chips still contain less on-chip memory than many of high-performance microprocessors (e.g., one of the highest-end FPGA chips, the Altera Stratix V, has a 6.4 MB embedded memory, whereas the i7 processor contains 8 MB cache memory, which is also significantly faster). Because current on-chip memory is barely sufficient to implement the networks we studied here, an external memory would be required for larger networks. Accessing an external memory while performing computation may greatly increase the energy consumption. The second limitation is related to the mapping of the algorithms that we considered on an FPGA chip. A direct mapping of the algorithm would be desirable for energy efficiency, but the needed abundance of fan-in/fan-out connections, memory, and computation would probably require a prohibitive amount of resources in an FPGA chip. These limitations can probably be overcome, but this would require a separate study.

Application-specific integrated circuits (ASIC) can be even more energy-efficiency—for example the Kerneltron (Genov & Cauwenberghs, 2003) and similar template-matching processors (Karakiewicz, Genov, & Cauwenberghs, 2007) or the full custom chip implementing the NeuFlow (Pham et al., 2012). However, ASIC chips have very limited programmability and are able to perform only the few specific tasks that the chips are designed for.

It is also interesting to discuss the performance of other conventional digital processors in the benchmarks we examined. Let us consider, for example, some implementations of SVMs classifying the MNIST digits with about 10^4 support vectors, which is roughly the number of vectors we need to achieve the best classification accuracy. As we have shown, the single-core configuration of Intel i7 takes about 8 ms to perform a classification, at an approximate cost of 20 mJ. The IBM chip, in contrast, requires 1 mJ for the longest classification times (500 ms) and 0.2 mJ for the average classification time (100 ms). We also used simulations to estimate the energy cost of the quad-core configuration of the Intel i7 processor that includes a level 3 (L3) cache memory. This configuration consumes slightly more energy (about 50 mJ) than the single core configuration. This is mostly due to the baseline

power dissipation of the additional circuitry implementing the large L3 cache, which is only partially compensated by the improvement in speed (0.4 ms) over the single-core configuration. Another architecture that could confer an advantage is ARMv7, which is a more energy-efficient yet slower microprocessor often used in mobile technologies. We quantified the energy cost of the ARMv7 and found that its energy consumption per classification was substantially higher—around 700 mJ. The main reason for this high consumption is that it takes more than 0.6 s to perform a single classification, and such a long time is penalized by baseline consumption (which increases linearly with classification time and takes up a large fraction of the total energy needed for a classification). Finally, we considered the recent Xeon Phi, which has a massively parallel architecture and is employed in high-performance computing applications. Because we do not have a simulator for the Phi, we could only indirectly estimate a lower bound for the energy consumption (see the appendix for more details). According to our estimate, a single classification requires only 0.2 μs and uses about 16 mJ, which would be significantly lower than the energy cost of the i7 and very close to the estimated lower limit of energy consumption (Hasler & Marr, 2013), but still larger than the consumption of the IBM chip. Notice, however, that both the classification time and the energy consumption of the Xeon Phi processor are very likely to be grossly underestimated, as they are simply derived from the peak performance of 100 Tflop/s. The estimates for the i7 and the ARMv7 are significantly more reliable because we derived them by simulating the processors.

To summarize, our results compellingly suggest that the neuromorphic approach is finally competitive in terms of energy consumption in useful real-world machine learning tasks and constitutes a promising direction for future scalable technologies. The recent success of deep networks for large-scale machine learning (Krizhevsky, Sutskever, & Hinton, 2012; Deng, Hinton, & Kingsbury, 2013) makes neuromorphic approaches particularly relevant and valuable. This will certainly be true for neuromorphic systems with synaptic plasticity, which will enable these devices to learn autonomously from experience. Learning is now available only in small neuromorphic systems (Mitra, Fusi, & Indiveri, 2009; Giulioni et al., 2011; Arthur & Boahen, 2006), but new VLSI technologies may allow us to implement it in large-scale neural systems too.

Appendix: Detailed Methods

A Neural Classifier.

A.1 Image Sets for Classification Benchmarks. We used three data sets in our study: MNIST, MNIST-back-image, and LaTeX. The MNIST data set consists of images of handrwitten digits (10 classes) (LeCun, Bottou, Bengio, & Haffner, 1998). The MNIST-back-image data set contains the same digits

Table 1: Data Set Properties.

Data Set	Image Size	Number of Classes	Training Set Size	Test Set Size
MNIST	28×28	10	60,000	10,000
MNIST-back-image	28×28	10	12,000	50,000
LaTeX	32×32	293	14,650	9376

of MNIST, but in this case the background of each pattern is a random patch extracted from a set of 20 black and white images downloaded from the Internet (Larochelle, Erhan, Courville, Bergstra, & Bengio, 2007). Patches with low pixel variance (i.e., containing little texture) are discarded. The LaTeX data set consists of distorted versions of 293 characters used in LaTeX document preparation system (Amit & Geman, 1997; Amit, 2002). All data sets consist of $l \times l$ pixel gray-scale images, and each of such pixel images is associated with one out of C possible classes. The size of the pixel images, the number of classes, and the sizes of the training and test sets depend on the data set (see Table 1).

Preprocessing. Every sample image was reshaped as a l^2-dimensional vector, and the average gray level of each component was subtracted from the data. The dimensionality of the resulting image vector was then reduced to 256 using PCA. To guarantee that all the selected components contributed uniformly to the patterns, we applied a random rotation to the principal subspace (see Raiko, Valpola, & LeCun, 2012). We denote by $N_{\text{in}} = 256$ the dimension of that subspace.

A.2 The Architecture of the Network and the Training Algorithm. We map the preprocessed N_{in}-dimensional vector image, $\mathbf{s}$, into a higher-dimensional space through the transformation

$$x_i = f(\mathbf{w}_i \cdot \mathbf{s}), \quad i = 1, \ldots, N,$$

where $\mathbf{w}_i$ is an N_{in}-dimensional sparse random vector and $f(\cdot)$ is a nonlinear function. This is the transformation induced by a neural network with N_{in} input units and N output units with activation function $f(\cdot)$. More succinctly,

$$\mathbf{x} = f(\mathbf{W}^T \mathbf{s}), \tag{A.1}$$

where $\mathbf{W}$ is a weight matrix of dimensions $N_{\text{in}} \times N$ formed by adjoining all the column weight vectors $\mathbf{w}_i$ and where $f(\cdot)$ acts componentwise, that is, $f(\mathbf{x}) \equiv (f(x_1), \ldots, f(x_{N_{\text{in}}}))^T$. The output of the random nonlinear transformation, $\mathbf{x}$, is used as the input to a linear N_C-class discriminant, consisting

of N_C linear functions of the type $y_j = \sum_{k=1}^{N} J_{jk} x_k$, with $j = 1, \ldots, C$. More compactly,

$$\mathbf{y} = \mathbf{J}\mathbf{x}, \tag{A.2}$$

where $\mathbf{y} = (y_1, \ldots, y_C)^T$, $\mathbf{J}$ is a $C \times N$ matrix and $\mathbf{x}$ is given by equation A.1. A pattern $\mathbf{x}$ is assigned to class C_j if $y_j(\mathbf{x}) > y_k(\mathbf{x})$ for all $j \neq k$. The elements of $\mathbf{J}$ are learned offline by imposing a 1-of-N_C coding scheme on the output: if the target class is j, then the target output $\mathbf{t}$ is a vector of length N_C where all components are zero except component t_j, which is 1. For the offline training of weights, we use the pseudoinverse, which minimizes the mean squared error of the outputs. This technique has been shown to be a good replacement for empirical minimization problems when the data set is embedded in a random high-dimensional space, which is our case (Huang, Zhu, & Siew, 2006; Rahimi & Recht, 2008; Tapson & van Schaik, 2013; Le et al., 2013).

B Neuromorphic Chip Implementation. The chip is composed of multiple identical cores, each of which consists of a neuromorphic circuit that comprises $n = 256$ axons, n neurons, and n^2 adjustable synapses (Merolla et al., 2011; Arthur et al., 2012; Merolla et al., 2014; see also Figure 1c). Each axon provides the inputs by feeding the spiking activity of one given neuron that may or not reside in the core. The incoming spiking activity to all n axons in a core is represented by a vector of activity bits $(A_1(t), \ldots, A_n(t))$ whose elements indicate whether the neurons associated with the incoming axons emitted a spike in the previous time step. The intersection of the n axons with the n neurons forms a matrix of programmable synapses. The weight of active synapses is determined by the type of axon and the type of neuron the synapse lies on. Specifically, each core can contain up to four different types of axon, labeled $G_j = \{1, 2, 3, 4\}$, whereas it can accommodate an unlimited number of neuron types, each of which having four associated synaptic weights $S_i = (S_i^1, \ldots, S_i^4)$. The strength of an active synapse connecting axon j with neuron i is $S_i^{G_j}$, that is, the axon type determines which weight to pick among the weights associated with neuron i. The net input received by neuron i at time step t is therefore $h_i(t) = \sum_{j=1}^{n} S_i^{G_j} B_{ij} A_j(t)$, where B_{ij} is 1 or 0 depending on whether the synapse between axon j and dendrite i is active or inactive.

At each time step, the membrane potential $V_i(t)$ of neuron i receiving input $h(t)$ is updated according to $V_i(t+1) = V_i(t) - \beta + h_i(t)$, where β is a constant leak. If $V_i(t)$ becomes negative after an update, it is clipped to 0. Conversely, when $V_i(t)$ reaches the threshold V_{thr}, the potential is reset to V_{reset} and the neuron emits a spike, which is sent through the neuron's axon to the target core and neuron. This design implies that each neuron

can connect to at most n neurons, which are necessarily in the same core. The initial voltage of each neuron was initialized by drawing randomly and with equal probability from a set of four evenly spaced values from V_{reset} to V_{thr}.

B.1 Signal-to-Rate Transduction. The input to the neuromorphic chip consists of a set of spike trains fed to the neurons of the input layer. To transform the vector signal $\mathbf{s}$ into spike trains, we first shifted the signal by $\bar{s} = 3\sigma$, where σ is the standard deviation across all signal components of all patterns. The shifted signal was then scaled by a factor ν_{sc} chosen to ensure moderate output rates in the RCN layer, and the result was linear-rectified to positive values. In short, the input rate ν_i associated with signal s_i is $\nu_i = \nu_{\text{sc}}[s_i + \bar{s}]_+$, $i = 1, \ldots, N_{\text{in}}$, where $[x]_+$ is x if $x > 0$, or 0 otherwise. The values ν_i were then used to generate regular spike trains with fixed interspike interval $1/\nu_i$.

B.2 Basic Architecture. The circuit is divided in two functional groups, or layers, each comprising several cores. The first functional group is the RCN layer, which computes the random nonlinear expansion in equation A.1. The second functional group computes the C-class discriminant $\mathbf{y} = \mathbf{J}\mathbf{x}$. The output of the classifier is just $\text{argmax}_j\, y_j$, where j runs over the C possible categories. The argmax operation was not computed by the chip but was determined offline by comparing the accumulated spike counts across all outputs. In the following, we describe the implementation of the two layers in more detail.

RCN layer. We first set the dimensionality of the input to the number of available axons per core: $N_{\text{in}} = n = 256$. A convenient choice for $\mathbf{W}$ is an $n \times N$ matrix where each column is a vector of zeros except for exactly $m < n$ nonzero entries, which are randomly placed and take a fixed integer value w. We took $m = 26$, which corresponds to a connectivity level of around 0.1. Lowering the connectivity has the advantage of decreasing energy costs by reducing the number of total spikes and active synapses, without affecting classification performance. The random expansion was mapped in the chip by splitting the matrix $\mathbf{W}^T$ into $\lceil N/n \rceil$ submatrices of size $n \times n$ and using each submatrix as the (Boolean) connectivity matrix B_{ij} of a core.

With this arrangement, each of the N neurons distributed among the $\lceil N/n \rceil$ cores receives a sparse and random linear combination of signals. Specifically, the average current received by each RCN is

$$h_i = \sum_{j=1}^{n} W_{ji} \nu_j, \quad i = 1, \ldots, N.$$

A zero-th order approximation of the firing rate of a general VLSI neuron receiving a current h_i is

$$r_i = \frac{[h_i - \beta]_+}{V_{\text{thr}} - V_{\text{reset}}}, \tag{B.1}$$

where V_{thr} is the threshold for spike emission and V_{reset} is the reset potential (Fusi & Mattia, 1999).

We chose the parameters w and β to meet two criteria. First, we required the fraction of RCNs showing any firing activity (i.e., the coding level f) to be around 0.25. This coding level is a good compromise between the need for discrimination and generalization, and it keeps finite-size effects at bay (Barak et al., 2013). Second, we required the distribution of activities across active RCNs to be sufficiently wide. Otherwise the information carried by the spiking activity of the RCNs is too imprecise to discriminate among patterns.

All the cores in the RCN layer receive exactly the same n-dimensional input signal.

Readout. The readout matrix **J** was trained offline and mapped to the chip architecture as follows.

• *Weight quantization.* Because the chip can hold only integer-valued synapses, we need to map the set of all components of **J** into an appropriate finite set of integers. We started clipping the synaptic weights within the bounds $(-4\sigma, 4\sigma)$, where σ is the standard deviation of the sample composed of all the components of **J**. We then rescaled the weights to a convenient magnitude $J_{\text{max}} = 28$ (see below) and rounded the weight values to the nearest integer.

• *Weight assignment.* The TrueNorth connectivity constraints dictate that each RCN can project to only one axon, meaning that there are at most $n = 256$ synaptic contacts available to encode the $C = 10$ weights, $J_{0i}, \ldots, J_{9i}$ associated with the ith RCN. We allocated 24 contacts per class and per axon (see Figure 6). Each of these 24 contacts was divided in 4 groups comprising 6 weights each, with values $1, 2, 4, -1, -2, -4$. This allowed us to represent any integer weight from -28 to 28 (each of the 4 groups encodes a maximum weight of 7, sign aside). To distribute any weight value w across the available synaptic contacts, we decomposed w in a sum of four terms, given by the integer division of w by 4, with the remainder spread evenly across terms (e.g., $19 = 4 + 5 + 5 + 5$). Each of these values was assigned to one group, represented in base 2, and mapped to a pattern of active-inactive synapses according to the weight associated with each axon-dendrite intersection. Positive and negative weights, as well as strong and weak weights, were balanced along a dendrite by changing the sign and order of the weights in the crossbar (see the alternating colors and saturations in Figure 6).

• *Negative threshold.* For the readout to work properly, the firing activity of readout neurons must be proportional to the linear sum of the inputs from the RCNs. This requires neurons to operate in the linear regime of their

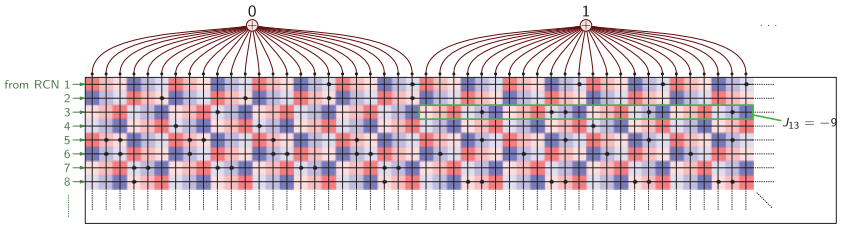

Figure 6: Implementation of the readout matrix in a core. The diagram represents the first 8 input lines and first 48 dendrites (2 output units) of a typical readout core. Under each axon-dendrite contact is a square that indicates the potential synaptic strength at the site. Color indicates whether the connection is excitatory (red) or inhibitory (blue), while the saturation level represents the absolute value of the synaptic strength, which can be 1, 2, or 4 (low, medium, and high saturation, respectively). Only the sites marked with a dot are active. The green frame highlights all the synaptic contacts allocated for an arbitrary weight of the readout matrix—in this case, $J_{13} = -9$, which is decomposed as the 4-term sum $-9 = -2 - 3 - 2 - 2 = -010_2 - 011_2 - 010_2 - 010_2$. Note that in this particular axon, the ordering of weights is $2^0, 2^1, 2^2$ (the right-most bit is the most significant).

dynamic range, a regime that can be enforced by lowering the threshold β_{out} of readout neurons. We set $\beta_{\text{out}} < 0$, which is equivalent to adding a constant positive current to each neuron. If the current-to-rate transduction function were the threshold-linear function of equation B.1, the baseline activity induced by this constant current would be $|\beta_{\text{out}}|/(V_{\text{thr}} - V_{\text{reset}})$ per readout neuron. The contribution of this background signal should be subtracted from the readout outputs if one wants to get the equivalent to equation A.2, although the step is unnecessary if one wishes only to compare output magnitudes (as we implicitly do in order to find the maximal output).

C Support Vector Machines. We trained SVMs to perform multiclass classifications based on a one-versus-all scheme, so that the number of output units coincides with the number of classes (as in the neural classifier). SVMs were evaluated using arc-cosine kernels, which mimic the computation of large feedforward networks with one or more layers of hidden nonlinear units (Cho & Saul, 2010). For our particular architecture, based on one hidden layer built with threshold-linear units, the kernel is $k(\mathbf{x}, \mathbf{y}) = \|\mathbf{x}\|\|\mathbf{y}\|J_1(\theta)$, where $J_1(\theta) = \sin\theta + (\pi - \theta)\cos\theta$ and θ is the angle between the inputs $\mathbf{x}$ and $\mathbf{y}$.

We considered three types of SVM. For the standard SVM we used the open library `libsvm` (Chang & Lin, 2011), which we patched to include the arccos kernel. The other two SVMs reduce the number of support vectors without sacrificing performance substantially. One of such algorithms is

`primalSVC`, which selects greedily the basis functions by optimizing the primal objective function (Keerthi, Chapelle, & DeCoste, 2006). The other method is based on the so-called cutting-plane subspace pursuit algorithm, which reduces the number of support vectors by using basis functions that, unlike standard SVMs, are not necessarily training vectors (Joachims & Yu, 2009). Such method is implemented in the library `SVMperf`. Unlike the other two classifiers, `SVMperf` used RBF kernels instead of arccos kernels.

D Estimation of the IBM Chip Energy Consumption. The energy consumption of the IBM chip was computed by using the estimates of Merolla et al. (2014), obtained by running a set of probabilistically generated complex recurrent networks (for more details, see Merolla et al., 2014, suppl. S7). The total energy consumption comprises the baseline energy (15.9 μW per core), the energy to emit spikes (109 pJ per spike), the energy needed to read active synapses (10.7 pJ per active synapse), and the energy necessary to update membrane potentials (1.2 pJ per neuron). We ignored the input-output energy needed to transmit spikes off chip and receive spikes on chip. These numbers provide a reasonable estimate of the energy consumption of systems with a conservative supply voltage of 0.775 V; most chips operate near or below this estimate. For a setup with 2^{14} RCNs, 26 dendrites per class, and 10 classes, the power was about 2.08 mW, 95% of which corresponds to the baseline power. The energy consumption depends also on the distance the spikes have to travel. In this respect, the estimates of Merolla et al. (2014) are highly conservative as they required spikes to travel long distances (21.3 cores away on average). In our case, each core communicates only with another core, and the cores can be arranged in such a way that the distance spikes need to travel is at most one.

D.1 Scaling of the Energy with the Number of Classes. The estimation was based on the energy cost of the simulated classifications of the MNIST data set and extrapolated to the designs required by an increasing number of classes. As the number of classes C increases, so does the number of readout neurons necessary to perform a classification and, therefore, so does the required number of readout cores. Specifically, if we assign s_c synaptic contacts per axon and per class, we will need a total of s_cC output lines. These output lines need to be connected to all the N neurons through the input lines of the readout cores . Because each readout core can accommodate 256 output lines, connected to 256 input lines, the total number of readout cores will be $\lceil N/256 \rceil \lceil s_cC/256 \rceil$ ($\lceil \cdot \rceil$ indicates the ceiling function). In principle the number of RCN cores will be simply $\lceil N/256 \rceil$. However, each RCN should project to $\lceil s_cC/256 \rceil$ cores, which implies that each RCN core must be cloned $\lceil s_cC/256 \rceil$ times due to the fan-out constraint—each RCN can project to only one core. The total number of cores is therefore $N_{\text{cores}} = 2\lceil N/256 \rceil \lceil s_cC/256 \rceil$, where the factor 2 accounts for the contributions of both the readout and the RCN cores.

The total number of spikes emitted was estimated from the reference value we got from the chip simulation (for 10 classes, $N = 2^{14}$, $s_c = 24$, and 500 ms of classification time), scaled appropriately for the new N_{cores}. More concretely, if we denote by n^0_{sp} the number of spikes emitted during our reference simulation, the number of emitted spikes in a general case is $n_{\text{sp}} = n^0_{\text{sp}} \lceil s_c C/256 \rceil (T/500)(N/2^{14})$, where T is the duration of the simulation in milliseconds. We chose this duration to be $T = 108$ ms, the average classification time of the chip implementation for the MNIST data set, when the spike difference is 80 spikes and yields only 0.1% less in performance than in the fixed-duration case (97.2% versus 97.3%). With T and the estimated values of N_{cores} and n_{sp}, it is straightforward to compute the energy consumption according to the values given in the previous paragraph.

E Energy Consumption in von Neumann Digital Machines

Configuration. The run time and power of microprocessors with von Neumann architectures were estimated with the recently developed simulators GEM5 (gem5.opt 2.0) (Binkert et al., 2011) and McPAT (ver. 1.2) (Li et al., 2009). For the estimation, we used an architecture configuration similar to that of the recent Intel Core i7 processors (http://www.intel.com/content/www/us/en/processors/core-i7-processor.html), which incorporate state-of-the-art CMOS technology. Specifically, we tested both single-core and quad-core configurations. The single-core configuration had a single x86_64, O3 core architecture at 2.66 GHz clock frequency, with 32 KB eight-way L1-i and 32 KB eight-way L1-d caches, 256 KB eight-way L2 cache, 64 B cache line size, and 8 GB DDR3 1600 DRAM. The quad-core configuration had four of the X86_64, O3 cores at the same clock frequency, with private 32 KB eight-way L1-i and private 32 KB 8-way L1-d caches, private 256 KB eight-way L2 cache, shared 8 MB eight-way L3 cache, 64B cache line size, and 8 GB DDR3 1600 DRAM. The SVM code for the quad-core experiment is rewritten with `pthreads` for multithread support. For both configurations, the channel length was 22 nm, HP type, using long channel if appropriate. VDD was 0.9 V, and thus slightly higher than the 0.775 V used for the IBM chip. However, could we use the same voltage in Intel i7 simulator, the energy consumption would be lower by a factor $(0.775/0.9)^2 = 0.74$. This 26% reduction would not change the main conclusions about the energy consumption gap between the IBM chip and the conventional von Neumann digital machines, which is two to three orders of magnitude.

E.1 Simulations. The benchmark was the test phase of the SVMs, already trained. Simulations showed that a modern microprocessor based on a von Neumann architecture takes 180.6 ms to evaluate 20 test patterns with 8424 SVs, while consuming 519.5 mJ (DRAM energy consumption not included). When we varied the number of support vectors from 61 to 8424, both the

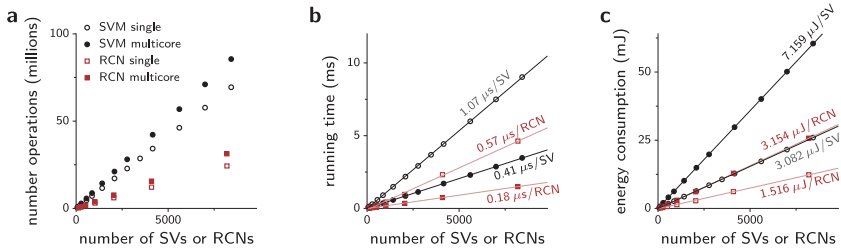

Figure 7: Simulation of a digital support vector machine. Number of operations (a), run time (b), and energy consumption (c) required to classify one test pattern from the MNIST data set, as a function of the number of SVs or RCNs used. Measures were obtained from a (single- or multicore configuration of) an Intel i7 processor and correspond to three different systems: a digital implementation of an SVM, both single and multithread, and a digital implementation of an RCN. The straight lines in panel c are a least-square fits.

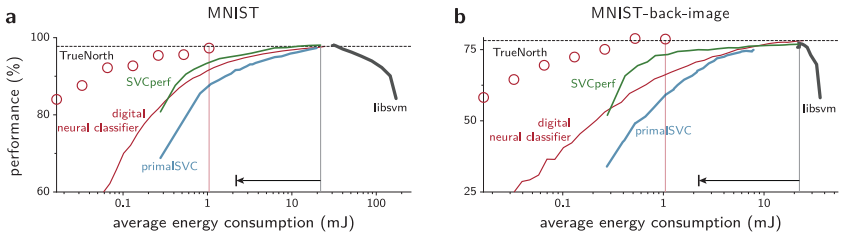

Figure 8: Performance versus energy consumption at a fixed classification time. Panels are like in Figures 4b and 4d, but classification time is now fixed at 500 ms rather than varying from trial to trial according to a stopping criterion.

run time and energy consumption grew proportionally to the number of SVs, while the power was roughly constant due to the fixed hardware configuration (see Figure 7).

To estimate how the energy used by von Neumann digital SVMs scales with the number of classes, we ran another set of simulations with an Intel i7 simulator, this time varying both the number of support vectors and the number of classes in the classification problem. This step was necessary to determine the overhead incurred when we increase the number of output units. For a given number of classes, the energy cost per support vector was estimated from the least-square fit of the energies against the number of support vectors.

Mobile processor. We also investigated the run time and energy consumption of a more energy-efficient but slower mobile microprocessor performing the same target workload. The architecture configuration was ARMv7, O3, single core, 1 GHz CPU clock frequency, 32 kB four-way L1i and 32 kB

four-way L1d caches, and 128 kB eight-way L2 cache, which is similar to the architecture of ARM Cortex-A9 (http://www.arm.com/products/processors/cortex-a/cortex-a9.php). The technology node (22 nm) and simulators were the same as in the experiment with the microprocessor mimicking Intel Core i7. For the benchmark code with the largest number of SVs, the task required $1.2 \cdot 10^{10}$ operations that took 6.35 s at a cost of 7.34 J.

Discussion on Intel Xeon Phi. Massively parallel architectures have gained a significant amount of attention to improve the throughput and power efficiency of the high-performance computing (HPC) technology, in response to the relatively stagnated improvement in clock frequency. The Xeon Phi coprocessor, recently developed by Intel, is one such effort (Chrysos & Engineer, 2012). It integrates more than 50 CPU cores together with L1/L2 caches, network-on-chips, GDDR memory controller, and PCIe interface. Each core supports up-to 4-thread in-order operation and the 512b SIMD VPU (vector processing unit). While the run time and energy-consumption of the coprocessor are highly dependent on the target workloads, several recent investigations quantified the performance and energy efficiency. In the high-performance configuration, the system integrating Xeon and Xeon Phi shows the throughput of 100 Tera floating-point operations (flop) per second, the power consumption of 72.9 kW, marking the energy efficiency of 0.74 nJ/flop (Chrysos & Engineer, 2012). The classification benchmark codes (with the largest number of SVs) require 0.02235 Gigaflop on the desktop processor configuration similar to Intel Core i7. At a first-order approximation, therefore, the Xeon and Xeon Phi-based system takes 0.2235 μs and uses 16.5 mJ per classification. This energy consumption seems significantly lower than the one of the Intel Core i7 and very close to its lower bound, which is approximately 3 mJ. However, one should keep in mind that the energy is grossly underestimated; not only did we ignore the energy needed for the RAM, but we also neglected the cost of the non-floating-point operations, which are approximately twice as many as the floating-point operations. For all these reasons, it is difficult to compare the energy consumption for the Xean Phi to the Intel Core i7. In any case, even for our very conservative energy consumption estimate, the IBM chip remains significantly more energy efficient.

Acknowledgments

This work was supported by DARPA SyNAPSE, Gatsby Charitable Foundation, Swartz Foundation, Grossman Foundation, and Kavli Foundation. We are grateful to the IBM team led by D. Modha for their assistance with the IBM chip simulator. In particular, we thank John Arthur and Paul Merolla for their help with the estimate of the chip energy consumption. We also thank Rajit Manohar for many useful comments on the manuscript. D.M. acknowledges support from the FP7 Marie Curie Actions of the European

Commission and the ANR-10-LABX-0087 IEC and ANR-10-IDEX-0001-02 PSL grants.

References

Amit, D. J., & Fusi, S. (1994). Learning in neural networks with material synapses. *Neural Comput.*, *6*(5), 957–982.

Amit, Y. (2002). *2D object detection and recognition: Models, algorithms, and networks*. Cambridge, MA: MIT Press.

Amit, Y., & Geman, D. (1997). Shape quantization and recognition with randomized trees. *Neural Comput.*, *9*(7), 1545–1588. 10.1162/neco.1997.9.7.1545

Arthur, J., & Boahen, K. (2006). Learning in silicon: Timing is everything. In Y. Weiss, B. Schölkopf, & J. Platt (Eds.), *Advances in neural information processing systems*, 18. Cambridge, MA: MIT Press.

Arthur, J. V., & Boahen, K. (2011). Silicon-neuron design: A dynamical systems approach. *IEEE Transactions on Circuits and Systems I: Regular Papers*, *58*(5), 1034–1043.

Arthur, J. V., Merolla, P. A., Akopyan, F., Álvarez, R., Cassidy, A., Chandra, S., . . . Manohar, R. S. (2012). Building block of a programmable neuromorphic substrate: A digital neurosynaptic core. In *Proceedings of the 2012 International Joint Conference on Neural Networks* (pp. 1–8). Piscataway, NJ: IEEE. 10.1109/ijcnn.2012.6252637

Barak, O., Rigotti, M., & Fusi, S. (2013). The sparseness of mixed selectivity neurons controls the generalization–discrimination trade-off. *J. Neurosci.*, *33*(9), 3844–3856. 10.1523/jneurosci.2753-12.2013

Benjamin, B. V., Gao, P., McQuinn, E., Choudhary, S., Chandrasekaran, A. R., Bussat, J. M., . . . Boahen, J. K., (2014). Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proceedings of the IEEE*, *102*, 699–716.

Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., . . . Wood, D. (2011). The GEM5 simulator. *ACM SIGARCH Computer Architecture News*, *39*(2), 1–7.

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory* (pp. 144–152). New York: ACM.

Buonomano, D. V., & Maass, W. (2009). State-dependent computations: Spatiotemporal processing in cortical networks. *Nat. Rev. Neurosci.*, *10*(2), 113–125. 10.1038/nrn2558

Butko, A., Garibotti, R., Ost, L., & Sassatelli, G. (2012). Accuracy evaluation of GEM5 simulator system. In *Proceedings of the 2012 7th International Workshop on Reconfigurable Communication-Centric Systems-on-Chip* (pp. 1–7). Piscataway, NJ: IEEE. 10.1109/recosoc.2012.6322869

Chang, C. C., & Lin, C. J. (2011). LIBSVM: A library for support vector machines. *ACM T. Intel. Sys. Techn.*, *2*, 1–27.

Chicca, E., Stefanini, F., & Indiveri, G. (2014). Neuromorphic electronic circuits for building autonomous cognitive systems. *Proceedings of the IEEE PP*, *99*, 1–22. 10.1109/jproc.2014.2313954

Cho, Y., & Saul, L. K. (2010). Large-margin classification in infinite neural networks. *Neural Comput.*, *22*(10), 2678–2697. 10.1162/neco_a_00018

Chrysos, G., & Engineer, S. P. (2012). Intel Xeon Phi coprocessor (codename knights corner). In *Proceedings of the 24th Hot Chips Symposium*. Piscataway, NJ: IEEE.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Mach. Learn.*, *20*(3), 273–297. 10.1007/bf00994018

Delbruck, T., & Lang, M. (2013). Robotic goalie with 3ms reaction time at 4 load using event-based dynamic vision sensor. *Frontiers in Neuroscience*, *7*(223). 10.3389/fnins.2013.00223

Deng, L., Hinton, G., & Kingsbury, B. (2013). New types of deep neural network learning for speech recognition and related applications: An overview. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing* (pp. 8599–8603). Piscataway, NJ: IEEE.

Eliasmith, C., Stewart, T. C., Choo, X., Bekolay, T., DeWolf, T., Tang, Y., . . . Rasmussen, D. (2012). A large-scale model of the functioning brain. *Science*, *338*(6111), 1202–1205. 10.1126/science.1225266

Farabet, C., Martini, B., Corda, B., Akselrod, P., Culurciello, E., & LeCun, Y. (2011). Neuflow: A runtime reconfigurable dataflow processor for vision. In *Workshops* (pp. 109–116). Piscataway, NJ: IEEE. 10.1109/cvprw.2011.5981829

Fusi, S. (2002). Hebbian spike-driven synaptic plasticity for learning patterns of mean firing rates. *Biol. Cybern.*, *87*(5), 459–470

Fusi, S., & Mattia, M. (1999). Collective behavior of networks with linear (VLSI) integrate-and-fire neurons. *Neural Comput.*, *11*(3), 633–652. 10.1162/089976699300016601

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: A guide to NP-completeness*. New York: Freeman.

Genov, R., & Cauwenberghs, G. (2003). Kerneltron: Support vector machine in silicon. *IEEE Transactions on Neural Networks*, *14*(5), 1426–1434.

Giulioni, M., Camilleri, P., Mattia, M., Dante, V., Braun, J., & Giudice, P. D. (2011). Robust working memory in an asynchronously spiking neural network realized with neuromorphic VLSI. *Front. Neurosci.*, *5*.

Goodfellow, I., Warde-Farley, D., Mirza, M., Courville, A., & Bengio, Y. (2013). Maxout networks. *ICML*, *28*, 1319–1327.

Gupta, S., Agrawal, A., Gopalakrishnan, K., & Narayanan, P. (2015). *Deep learning with limited neural precision*. arXiv:1502.02551

Han, J., & Orshansky, M. (2013). Approximate computing: An emerging paradigm for energy-efficient design. In *Proceedings of the 18th IEEE European Test Symposium* (pp 1–6). Piscataway, NJ: IEEE.

Hasler, J., & Marr. B. (2013). Finding a roadmap to achieve large neuromorphic hardware systems. *Front. Neurosci.*, *7*. 10.3389/fnins.2013.00118

Huang, G. B., Zhu, Q. Y., & Siew, C. K. (2006). Extreme learning machine: Theory and applications. *Neurocomputing*, *70*(1), 489–501.

Hussain, S., & Basu, A. (2016). Multiclass classification by adaptive network of dendritic neurons with binary synapses using structural plasticity. *Frontiers in Neuroscience*, *10*(113). 10.3389/fnins.2016.00113

Indiveri, G., Linares-Barranco, B., Hamilton, T. J., Schaik, A. V., Etienne-Cummings, R., Delbruck, T., . . . Renaud, S. (2011). Neuromorphic silicon neuron circuits. *Front. Neurosci.*, *5*. 10.3389/fnins.2011.00073

Jaeger, H., & Haas, H. (2004). Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, *304*(5667), 78–80. 10.1126/science.1091277

Joachims, T., & Yu, C. N. (2009). Sparse kernel SVMs via cutting-plane training. *Mach. Learn.*, *76*(2–3), 179–193. 10.1007/s10994-009-5126-6

Karakiewicz, R., Genov, R., & Cauwenberghs, G. (2007). 480-GMACS/mW resonant adiabatic mixed-signal processor array for charge-based pattern recognition. *IEEE Journal of Solid-State Circuits*, *42*(11), 2573–2584.

Keerthi, S. S., Chapelle, O., & DeCoste, D. (2006). Building support vector machines with reduced classifier complexity. *J. Mach. Learn. Res.*, *7*, 1493–1515.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Bvrges, L. Bottou, & K. Weinberger (Eds.), *Advances in neural information processing systems* (pp. 1097–1105). Red Hook, NY: Curran.

Lan, G., Sartori, P., Neumann, S., Sourjik, V., & Tu, Y. (2012). The energy-speed-accuracy trade-off in sensory adaptation. *Nat. Phys.*, *8*(5), 422–428.

Larochelle, H., Erhan, D., Courville, A., Bergstra, J., & Bengio, Y. (2007). An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th International Conference on Machine Learning* (pp. 473–480). New York: ACM. 10.1145/1273496.1273556

Le, Q., Sarlós, T., & Smola, A. (2013). Fastfood approximating kernel expansions in loglinear time. In *Proceedings of the International Conference on Machine Learning*. JMLR.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proc. IEEE*, *86*(11), 2278–2324. 10.1109/5.726791

Li, S., Ahn, J. H., Strong, R. D., Brockman, J. B., Tullsen, D. M., & Jouppi, N. P. (2009). McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture* (pp. 469–480). Piscataway, NJ: IEEE.

Livi, P., & Indiveri, G. (2009). A current-mode conductance-based silicon neuron for address-event neuromorphic systems. *Proceedings of the IEEE International Symposium on Circuits and Systems* (pp. 2898–2901). Piscataway, NJ: IEEE.

McDonnell, M. D., Boahen, K., Ijspeert, A., & Sejnowski, T. J. (2014). Engineering intelligent electronic systems based on computational neuroscience. *Proceedings of the IEEE*, *102*(5), 646–651.

Mead, C. (1989). *Analog VLSI implementation of neural systems*. Reading, MA: Addison-Wesley.

Merolla, P., Arthur, J., Akopyan, F., Imam, N., Manohar, R., & Modha, D. S. (2011). A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm. In *Proceedings of the Custom Integrated Circuits Conference* (pp. 1–4). Piscataway, NJ: IEEE.

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., . . . Modha, D. S. (2014). A million spiking-neuron integrated circuit with

a scalable communication network and interface. *Science*, *345*(6197), 668–673. 10.1126/science.1254642

Mitra, S., Fusi, S., & Indiveri, G. (2009). Real-time classification of complex patterns using spike-based learning in neuromorphic VLSI. *IEEE Transactions on Biomedical Circuits and Systems*, *3*(1), 32–42

Neftci, E., Das, S., Pedroni, B., Kreutz-Delgado, K., & Cauwenberghs, G. (2014). Event-driven contrastive divergence for spiking neuromorphic systems. *Frontiers in Neuroscience*, *7*(272). 10.3389/fnins.2013.00272

Neftci, E. O., Pedroni, B. U., Joshi, S., Al-Shedivat, M., & Cauwenberghs, G. (2015). *Unsupervised learning in synaptic sampling machines*. CoRR abs/1511.04484. http://arxiv.org/abs/1511.04484

O'Connor, P., Neil, D., Liu, S. C., Delbruck, T., & Pfeiffer, M. (2013). Real-time classification and sensor fusion with a spiking deep belief network. *Frontiers in Neuroscience*, *7*(178). 10.3389/fnins.2013.00178

Painkras, E., Plana, L., Garside, J., Temple, S., Galluppi, F., Patterson, C., . . . Furber, S. B. (2013). SpiNNaker: A 1-W 18-core system-on-chip for massively-parallel neural network simulation. *IEEE Journal of Solid-State Circuits*, *48*(8), 1943–1953.

Pfeil, T., Grübl, A., Jeltsch, S., Müller, E., Müller, P., Petrovici, M. A., . . . Meier, K. (2013). Six networks on a universal neuromorphic computing substrate. *Frontiers in Neuroscience*, *7*(11). 10.3389/fnins.2013.00011

Pham, P. H., Jelaca, D., Farabet, C., Martini, B., LeCun, Y., & Culurciello, E. (2012). Neuflow: Dataflow vision processing system-on-a-chip. In *Proceedings of the 2012 IEEE 55th International Midwest Symposium on Circuits and Systems* (pp. 1044–1047). Piscataway, NJ: IEEE. 10.1109/mwscas.2012.6292202

Rahimi, A., & Recht, B. (2008). Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In D. Koller, D. Schuurmans, Y. Bengio, & L. Bottou (Eds.), *Advances in neural information processing systems* (pp. 1313–1320). Cambridge, MA: MIT Press.

Raiko, T., Valpola, H., & LeCun, Y. (2012). Deep learning made easier by linear transformations in perceptrons. In *Proceedings of the Conference on AI and Statistics* vol. 22 (pp. 924–932). http://jmlr.csail.mit.edu/proceedings/papers/v22/raiko12/raiko12.pdf

Rangan, V., Ghosh, A., Aparin, V., & Cauwenberghs, G. (2010). A subthreshold aVLSI implementation of the Izhikevich simple neuron model. In *Proceedings of the Engineering in Medicine and Biology Society 2010 Annual International Conference* (pp. 4164–4167). Piscataway, NJ: IEEE. 10.1109/iembs.2010.5627392

Sohn, K., Zhou, G., Lee, C., & Lee, H. (2013). Learning and selecting features jointly with point-wise gated Boltzmann machines. In *Proceedings of the 30th International Conference on Machine Learning* (pp. 217–225). JMLR.

Sompolinsky, H. (1986). Neural networks with non-linear synapses and static noise. *Phys. Rev. A*, *34*, 2571.

Steinwart, I., & Christmann, A. (2008). *Support vector machines*. Berlin: Springer.

Stromatias, E., Neil, D., Galluppi, F., Pfeiffer, M., Liu, S. C., & Furber, S. (2015). Scalable energy-efficient, low-latency implementations of trained spiking deep belief networks on spiNNaker. In *Proceedings of the International Joint Conference on Neural Networks* (pp. 1–8). Piscataway, NJ: IEEE.

Tang, H., Buia, C., Madhavan, R., Crone, N. E., Madsen, J. R., Anderson, W. S., & Kreiman, G. (2014). Spatiotemporal dynamics underlying object completion in human ventral visual cortex. *Neuron*, *83*(3), 736–748.

Tapson, J., & van Schaik, A. (2013). Learning the pseudoinverse solution to network weights. *Neural Netw.*, *45*, 94–100.

Vapnik, V., Golowich, S. E., & Smola, A. (1997). Support vector method for function approximation, regression estimation, and signal processing. In M. I. Jordan & T. Petsche (Eds.), *Advances in neural information processing systems, 9* (pp. 281–287). Cambridge, MA: MIT Press.

Xi, S. L., Jacobson, H., Bose, P., Wei, G. Y., & Brooks, D. (2015). Quantifying sources of error in mcPAT and potential impacts on architectural studies. In *Proceedings of the 21st International Symposium on High Performance Computer Architecture* (pp. 577–589). Piscataway, NJ: IEEE. 10.1109/hpca.2015.7056064

Received November 4, 2015; accepted June 13, 2016.